

Tecnología NoSQL COSMOS DB

Prof. María Roxana Martínez

Alumnos:

Tordoya Gerardo

Riviello Eugenio

Orsingher Pamela

Pereiro Rodrigo

Universidad Abierta Interamericana

TRABAJO PRÁCTICO DE INVESTIGACIÓN NOSQL

Dada la experiencia laboral y/o académica adquirida en su entorno de trabajo o bien a lo largo de la cursada, se solicita desarrollar un Trabajo Práctico de Investigación orientado a un motor o al concepto de NoSQL de su elección.

El mismo deberá contar con al menos 6 aspectos de la materia como alcance del trabajo (Introducción, trabajos relacionados, ventajas, desventajas, etc.).

Para ello se le solicita un archivo en formato PDF con los siguientes contenidos:

- Portada (con datos principales del alumno/a);
- Índice de temas propuestos;
- Contenidos del tema tratado;
- Conclusión/reflexión del estudiante y cómo se integran los contenidos de la materia;
- Referencias bibliográficas utilizadas.

Para este trabajo, no deben superar las 45 hojas.

Como proceso final, se deberán grabar y subir la presentación a un enlace privado en YouTube (su URL deberá figurar en el archivo de la carpeta entregada).

ÍNDICE

Introducción	3
Conceptos básicos de DocumentDB	5
Propiedades anidadas maestro-detalle	8
Cuándo usar una base de datos de documentos	10
¿Por qué DocumentDB?	11
Consultas enriquecidas en SQL	12
Desarrollo del lado del cliente	13
Desarrollo del lado del servidor	14
Escalabilidad	15
Consistencia	16
Costos	17
Notas sobre las estadísticas	18
Capacidad de proceso elevada	18
Datos jerárquicos	19
Redes y relaciones complejas	19
Esquema fluido	21
Microservicios	21
Consideraciones críticas	22
Transacciones con varias relaciones que apuntan a la misma entidad	22
Transacciones que necesitan una gran coherencia en todo el conjunto de datos	22
Notas sobre UX (experiencia de usuario)	23
Primeros pasos (el Data-End)	23
El desarrollo (Back-End & Front-End)	26
La aplicación	28
Pasos finales (UX)	29
La frutilla del postre	30
Conclusión	33
Link Video	33
Fuentes (Bibliografía)	34

INTRODUCCIÓN

Azure Cosmos DB de Microsoft es un servicio de datos multimodal distribuido globalmente que le permite escalar de forma elástica el rendimiento y el almacenamiento en cualquier número de regiones geográficas con baja latencia, alta disponibilidad y coherencia. El motor de base de datos de Azure Cosmos DB admite de forma nativa el dialecto SQL de DocumentDB, la API de MongoDB, la API de Gremlin (grafos) y las API de almacenamiento de Tablas de Azure.

Nosotros nos centraremos en la parte DocumentDB de Cosmos DB, que es la plataforma de base de datos de documentos NoSQL de Microsoft que se ejecuta en Azure.

SQL son las siglas de Structured Query Language y es un acrónimo asociado a las bases de datos relacionales, ya que es el lenguaje estándar y más utilizado para trabajar con bases de datos relacionales. Al usar el término NoSQL, lo que realmente queremos expresar es que nos estamos refiriendo a bases de datos no relacionales. Las bases de datos no relacionales abandonan muchos de los conceptos que utilizan las bases de datos relacionales tradicionales, en particular, cómo se estructuran y organizan los datos en un formato tabular donde cada columna tiene un tipo de datos específico: Las tablas contienen filas, todas con el mismo número de columnas, cada una de las cuales representa un registro.

Existen varios tipos de bases de datos no relacionales, como almacenes de clave-valor, almacenes de columnas y bases de datos de grafos y documentos. Azure Table Storage es un almacén de clave-valor, Cassandra es un almacén de columnas y Neo4j es una base de datos de grafos. MongoDB y DocumentDB son bases de datos de documentos. Aunque existen diferencias clave entre los distintos tipos de bases de datos NoSQL, también tienen características en común. Las bases de datos NoSQL están diseñadas para escalar horizontalmente y no solo hacia arriba, lo que significa que, si bien las bases de datos relacionales se pueden escalar simplemente agregando recursos de hardware adicionales, es más difícil escalar horizontalmente una base de datos relacional: No es fácil distribuir los datos entre varias particiones una vez que comienza a tocar el techo en la CPU, el disco y la memoria, y ya no puede escalar hacia arriba.

A diferencia de las bases de datos relacionales, las bases de datos NoSQL están diseñadas para escalar tanto como sea necesario, lo que hace que sea mucho más fácil alcanzar la escala de Internet. No están sujetos a una estructura tabular y columnar rígida, por lo que los datos se pueden organizar sin esquemas. Libre de esquemas significa que ya no necesitamos lo que tradicionalmente consideraríamos como registros en la base de datos. Somos libres de almacenar información de una manera que puede no ser la misma para cada registro, por lo que podríamos tener diferentes números y tipos de columnas, no solo con diferentes valores, sino también con diferentes tipos de datos.

Por diseño, las bases de datos NoSQL se crearon para simplificar, en términos de qué tipo de información se puede escribir. No son tan sólidas como las bases de datos relacionales tradicionales como SQL Server u Oracle, y ni siquiera intentan reflejar o proporcionar la

funcionalidad completa de estas. Esta es la razón por la que pueden superar a las bases de datos relacionales a gran escala, escalando fácilmente y no solo hacia arriba.

Debido a que las bases de datos NoSQL no tienen esquemas, son escalables y fáciles de usar, nos brindan más opciones para nuestras aplicaciones que las que teníamos antes. Las bases de datos relacionales tradicionales definitivamente llegaron para quedarse. Sin embargo, ya no tienen un monopolio firme como opción predeterminada cuando necesitamos elegir una base de datos para nuestras necesidades de desarrollo de aplicaciones.

DocumentDB es una base de datos no relacional de documentos que ha sido diseñada para objetos (documentos) de notación de objetos de JavaScript (JSON) con elementos etiquetados que pueden serializarse y deserializarse¹, generalmente sin esquema. Un documento en términos NoSQL se refiere a un objeto JSON y no debe confundirse con el significado típico de la palabra "documento", que tradicionalmente se refiere a PDF, Word y otros formatos de archivos de documentos similares.

Exploraremos (a continuación, brevemente) las características de DocumentDB y cómo se compara con las bases de datos relacionales tradicionales.

A lo largo de este trabajo, los términos Cosmos DB y DocumentDB se usan indistintamente ya que Cosmos DB se originó a partir de DocumentDB.

¹ En ciencias de la computación, la serialización (o marshalling en inglés) consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML o JSON, entre otros. La serie de bytes o el formato pueden ser usados para crear un nuevo objeto que es idéntico en todo al original, incluido su estado interno (por tanto, el nuevo objeto es un clon del original). La serialización es un mecanismo ampliamente usado para transportar objetos a través de una red, para hacer persistente un objeto en un archivo o base de datos, o para distribuir objetos idénticos a varias aplicaciones o localizaciones.

CONCEPTOS BÁSICOS DE DOCUMENTDB

DocumentDB, como su nombre lo indica, almacena datos como "documentos", que en realidad son objetos JSON. En una base de datos relacional, los registros se almacenan como filas en una tabla con columnas específicas utilizando un esquema definido. Sin embargo, en un almacén de documentos, los registros son documentos y cada uno contiene propiedades específicas, sin esquema. Para comprender las diferencias entre una base de datos relacional y una de documentos, consideremos la siguiente tabla:

Tabla 1: Principales diferencias entre una base de datos relacional y documental

Base de datos relacional	Base de datos de documentos
Filas	Documentos
Columnas y tipos de datos	Propiedades
Esquemas definidos y tipificados	Sin esquemas
Altamente normalizado	Mayormente desnormalizado
Escala verticalmente (más hardware)	Escala horizontalmente

Una tabla de base de datos relacional tiene una estructura fija y para realizar cambios en una tabla (como agregar una nueva columna, cambiar una columna específica para permitir valores NULL o cambiar un tipo de datos) es necesario modificar el esquema de la tabla, lo que puede crear efectos secundarios para los datos existentes ya almacenados en la tabla.

Debido a que DocumentDB no tiene esquemas, no está sujeto a las restricciones que tienen las bases de datos relacionales. En el mundo actual, donde el desarrollo de aplicaciones y software está asociado con esquemas de datos que evolucionan con frecuencia, las bases de datos de documentos son una gran combinación. Veamos cómo se almacenarían algunos datos de ejemplo en una tabla relacional tradicional y cómo se almacenarían en una base de datos de documentos.

Tabla 2: Datos de ejemplo almacenados en una tabla de base de datos relacional

Nombre	Apellido	Edad	Activo
Juan	Pérez	44	Activo
Cristiano	Ronaldo	33	NULL
Peter	Pan	67	Retirado

Como podemos ver, los datos están estructurados en un formato tabular con todos los registros teniendo el mismo número de columnas (tenga en cuenta que se utilizó una columna Edad por simplicidad: en una base de datos real, usaría una columna de fecha de nacimiento).

Considere el registro resaltado en amarillo. Este registro tiene valores en las columnas Nombre, Apellido y Edad, pero la columna Activo no tiene ningún valor; por lo tanto, se establece en NULL. Los demás registros, a diferencia del amarillo, tienen valores en todas sus columnas. Aunque el

formato tabular puede acomodar registros que no necesariamente tienen valores en el mismo número de columnas (al usar valores NULL en columnas que no tienen un valor), todavía requiere que esos registros tengan esa columna, ya que el esquema lo exige. Si tuviéramos que acomodar estos mismos datos en una base de datos de documentos no relacionales como DocumentDB, se vería de la siguiente manera:

Tabla 3: Datos de ejemplo almacenados en una base de datos de documentos no relacionales

Documento	Datos del documento
1	{ "Nombre": "Juan", "Apellido": "Pérez", "Edad": 44, "Activo": "Active" }
2	{ "Nombre": "Cristiano", "Apellido": "Ronaldo", "Edad": 33 }
3	{ "Nombre": "Peter", "Apellido": "Pan", "Edad": 67, "Activo": "Retirado" }

Solo con fines ilustrativos, los datos de la tabla anterior se muestran en un formato tabular, que se asemeja al formato que utiliza una tabla de base de datos relacional. Sin embargo, una base de datos de documentos no almacena los datos en un formato tabular, sino como una representación binaria de los datos JSON.

Lo que se conoce como una tabla en el contexto de una base de datos relacional se llama una colección de documentos en una base de datos de documentos o, simplemente, una colección. Tenga en cuenta que las principales diferencias entre las bases de datos de documentos relacionales y no relacionales son que, en las bases de datos de documentos no relacionales, los registros se denominan documentos y solo almacenan valores para las propiedades que se utilizan (que corresponderían a columnas en una tabla relacional). Es decir, no es necesario representar las propiedades NULL dentro de los documentos, ya que no se imponen mediante un esquema.

Además, una base de datos de documentos no relacionales permite que los documentos almacenen subpropiedades, que no se pueden almacenar explícitamente dentro de una tabla relacional. Un ejemplo de esto sería reemplazar las propiedades Nombre y Apellido con una propiedad NombreCompleto con dos subpropiedades denominadas Nombre y Apellido.

Tabla 4: Un documento con subpropiedades

Documento	Datos del documento
2	<pre>{ "NombreCompleto": { "Nombre": "Cristiano", "Apellido": "Ronaldo" }, "Edad": 31 }</pre>

Este ejemplo muestra cómo, al tener subpropiedades, el documento no está normalizado con respecto a otros documentos presentes en la misma colección, pero una base de datos de documentos puede acomodar esto perfectamente. Esto significa que cada documento puede tener propiedades y subpropiedades que pueden no estar necesariamente presentes en otros documentos, lo que permite un alto grado de flexibilidad y desnormalización de los datos almacenados.

PROPIEDADES ANIDADAS MAESTRO-DETALLE

En las bases de datos no relacionales, las relaciones maestro-detalle se definen mediante propiedades anidadas. En una base de datos relacional, la única forma de lograr esto es tener una relación maestro-detalle entre dos tablas, compartiendo un campo común entre ellas. Consideremos este ejemplo de una relación maestro-detalle dentro de una base de datos relacional:

Tabla 5: Una relación típica de base de datos relacional maestro-detalle

Nombre	Apellido	Edad	Activo	JugadorId
Juan	Pérez	44	Activo	1
Cristiano	Ronaldo	33	NULL	2
Peter	Pan	67	Retirado	3

JugadorId	NombreEquipo
2	Real Madrid
2	Manchester United
1	Mars Galactic Soccer
3	Fantasy FC

En una base de datos relacional, como en el ejemplo anterior, la única forma de indicar que Cristiano Ronaldo ha jugado tanto con el Real Madrid como con el Manchester United, es tener una segunda tabla (de detalles) que contenga los nombres de ambos equipos y agregar una columna adicional a la tabla maestra con un JugadorId único que identifica el registro. Esta columna JugadorId también existirá en la tabla de detalle y será responsable de vincular los registros de detalle con el registro correspondiente en la tabla maestra. Esta es una relación de base de datos relacional maestro-detalle estándar.

En una base de datos de documentos, esta relación maestro-detalle se representa como una propiedad con propiedades anidadas.

Exploremos cómo se vería esto para el registro del jugador Cristiano Ronaldo:

Tabla 6: Un documento con propiedades anidadas de maestro-detalle

Documento	Datos del documento
2	<pre>{ "Nombre": "Cristiano", "Apellido": "Ronaldo", "Edad": 31, "Equipos": [{ "Equipo": "Real Madrid" }, { "Equipo": "Manchester United" }] }</pre>

Podemos ver que los equipos que se almacenarían en la tabla de detalles en una base de datos relacional tradicional ahora se almacenan dentro de una matriz denominada Equipos, en la que cada equipo es un objeto JSON anidado que contiene una subpropiedad denominada Equipo.

Básicamente, la relación maestro-detalle ahora se describe como una matriz de objetos JSON. Sin embargo, el hecho de que la relación maestro-detalle se pueda describir fácilmente como una matriz de objetos JSON no significa que siempre deba describirse como tal.

CUÁNDO USAR UNA BASE DE DATOS DE DOCUMENTOS

El poder y la belleza de las bases de datos no relacionales es que usted es libre de implementar y representar la relación entre registros maestro-detalle de la manera que mejor se adapte a sus requisitos comerciales y de aplicación.

Tampoco es raro en las bases de datos de documentos repetir algunos datos para que cada documento tenga los datos que necesita sin tener que localizar otros documentos. Si los datos se repiten demasiado, puede optar por organizar los datos repetidos en diferentes documentos, de forma similar a una base de datos relacional tradicional. En cualquier caso, puede organizar la estructura de sus documentos JSON según lo que mejor se adapte a sus requisitos y aplicaciones.

En esencia, una base de datos de documentos le brinda la libertad de modelar sus datos de la manera que mejor se adapte a sus necesidades. A pesar de esta flexibilidad, es importante comprender que las bases de datos de documentos son más adecuadas cuando se trabaja con datos que se pueden organizar en documentos jerárquicos ricos que pueden ser casi totalmente autónomos. Si se encuentra modelando una base de datos que contiene muchos documentos relacionados o documentos con una estructura plana², esta es una clara señal de que una base de datos de documentos probablemente no sea la mejor opción para su aplicación.

Cuando necesita una base de datos escalable, una base de datos de documentos es una excelente opción. Las razones principales por las que las bases de datos de documentos pueden escalar horizontalmente son que no imponen reglas complejas o rígidas sobre los datos y que son simples y sencillas por diseño³. Por otro lado, las bases de datos relacionales son más adecuadas para manejar requisitos complejos que no necesariamente tienen que escalar horizontalmente.

10

² Un archivo plano es una colección de datos almacenados en una base de datos bidimensional en la que cadenas de información similares pero discretas se almacenan como registros en una tabla. Uno de los ejemplos de archivos sin formato más destacados es un archivo de valores separados por comas (CSV).

³ ¿Por qué NoSQL es mejor en "escalamiento horizontal" que SGBDR? La ventaja de NoSQL sobre lo Relacional es el escalado horizontal, también conocido como fragmentación. Teniendo en cuenta que los 'documentos' NoSQL son una especie de objeto 'autocontenidos' (atómicos), los objetos pueden estar en diferentes servidores sin preocuparse por unir filas de varios servidores, como es el caso del modelo relacional (y ésta es la razón de que cláusula JOIN sea la marca distintiva de la diferencia entre relacional y NoSQL).

¿POR QUÉ DOCUMENTDB?

DocumentDB es la API de base de datos de documentos altamente escalable de Microsoft que se ejecuta en Azure Cosmos DB. Aunque tiene todas las características de una base de datos de documentos típica, también tiene características que no están disponibles en ninguna otra base de datos de documentos. exploremos estas características.

Con DocumentDB, a diferencia de otras bases de datos de documentos donde necesita definir índices explícitamente, todas las propiedades se indexan automáticamente tan pronto como el documento se agrega a la base de datos. Esto le permite buscar cualquier propiedad dentro de la jerarquía del documento, por muy anidada que esté.

Además, los documentos se pueden buscar utilizando una versión especial de SQL que cualquier persona con experiencia en SQL puede comprender y relacionar fácilmente de una manera intuitiva: este es el propio dialecto de SQL de DocumentDB.

Dado que DocumentDB se ejecuta en Azure Cosmos DB, proporciona un entorno del lado del servidor en el que se puede ejecutar código JavaScript que puede actualizar varios documentos con un procesamiento transaccional completo. Esta es una manera excelente y fácil de garantizar la coherencia de los datos entre varios documentos.

Además, DocumentDB permite un rendimiento ajustable para los requisitos de su aplicación, como mejora del rendimiento, indexación y consistencia. El rendimiento se puede escalar hacia arriba o hacia abajo instantáneamente cambiando el nivel de rendimiento a través de Azure Portal.

Aunque DocumentDB indexa automáticamente cada propiedad, aún puede ajustar el sistema para excluir cualquier propiedad o documento que no necesite indexarse, lo que incluso podría ayudar a mejorar el rendimiento en escenarios muy específicos.

Aunque DocumentDB es compatible con la consistencia sólida tradicional y eventual (una forma ligeramente diferente asociada con los sistemas de datos distribuidos), también proporciona dos opciones adicionales para brindarle un mayor control sobre las ventajas y desventajas entre el rendimiento y la consistencia.

Todas estas funcionalidades están muy bien empaquetadas como una solución de “plataforma como servicio” (PaaS), escalable masivamente, completamente administrada, y que es muy fácil de configurar y comenzar: No hay nada que instalar, ni sistema operativo ni actualizaciones que administrar, ni réplicas que configurar.

A través de Azure Portal, puede comenzar a usar DocumentDB en cuestión de minutos usando un navegador y teniendo una suscripción de Azure.

Exploremos algunas de estas características con algunos detalles adicionales.

CONSULTAS ENRIQUECIDAS EN SQL

Una de las mejores características de DocumentDB es que su lenguaje de consulta nativo es muy similar a SQL. Para aquellos que desarrollan con .NET, también hay un proveedor LINQ.

Aunque las consultas enriquecidas de DocumentDB están escritas en SQL, están profundamente arraigadas en la semántica JSON y JavaScript: Le permiten consultar matrices y datos anidados jerárquicos dentro de documentos y también compartir proyecciones personalizadas de los resultados de sus consultas.

Veamos un ejemplo:

Tabla 7: Una consulta SQL enriquecida de DocumentDB

```
-- Consulta enriquecida SQL.  
SELECT h.Nombre, h.Apellido  
FROM Jugadores AS j  
JOIN h IN j.Hijos  
WHERE j.Edad = 33
```

En este ejemplo de consulta SQL enriquecida, la cláusula JOIN básicamente permite que DocumentDB itere a través de todos los elementos secundarios (propiedades) anidados dentro del documento Jugadores. La cláusula WHERE filtra comprobando los documentos que tienen un valor de propiedad Edad igual a 31.

Observe que la notación punteada se utiliza para hacer referencia a las propiedades dentro del documento. La notación punteada se puede anidar hasta dónde llega la jerarquía del documento.

Debido a que en la cláusula SELECT estamos seleccionando dos propiedades en lugar de SELECT *, DocumentDB proyecta un nuevo objeto JSON que solo contiene las propiedades que se consultan, en lugar de todo el documento como resultado.

Una vez que se inserta un documento en DocumentDB, se puede buscar prácticamente al instante, ya que se indexa automáticamente. Este comportamiento de indexación se puede ajustar. Sin embargo, generalmente no hay necesidad.

DESARROLLO DEL LADO DEL CLIENTE

DocumentDB proporciona varios SDK, que permiten una fácil integración desde su plataforma de desarrollo preferida: .NET, Node.js, JavaScript, Java y Python.

No es diferente a otras plataformas, ya que también proporciona una API REST/HTTP que se puede usar siempre que los encabezados de la solicitud HTTP contengan información de autenticación válida y la solicitud apunte al recurso DocumentDB HTTP correcto.

Si no hay un SDK disponible para su plataforma de desarrollo, puede usar la API REST/HTTP. Sin embargo, también debe ponerse en contacto con el soporte técnico de Azure, informarles qué plataforma está utilizando y proporcionarles sus comentarios. Con suerte, pueden tener en cuenta su opinión e incluir un SDK para su plataforma de desarrollo en su hoja de ruta. El equipo de Azure es muy activo y está comprometido con la plataforma, por lo que, si no hay un SDK para su plataforma, es probable que nadie lo haya pedido todavía.

DESARROLLO DEL LADO DEL SERVIDOR

DocumentDB es un entorno de espacio aislado de servidor que le brinda la capacidad de ejecutar la lógica interna, donde residen los datos. La lógica del lado del servidor en DocumentDB se puede envolver en procedimientos almacenados, disparadores, y funciones definidas por el usuario. Todo esto es sorprendentemente familiar si se ha trabajado con sistemas de bases de datos relacionales como Oracle y SQL Server.

Sin embargo, existe una sutil diferencia entre la lógica del lado del servidor escrita en DocumentDB y la lógica del lado del servidor escrita con bases de datos relacionales tradicionales. En DocumentDB, la lógica del lado del servidor está escrita en JavaScript en lugar de SQL, lo que lo convierte en el compañero perfecto para manejar objetos JSON. DocumentDB adopta JavaScript como una especie de SQL moderno al admitir la ejecución de secuencias de comandos transaccionales de forma nativa dentro del motor de la base de datos.

Debido a que DocumentDB es un servicio totalmente alojado, no puede permitir que los scripts que funcionan mal se ejecuten indefinidamente, ya que esto podría poner en riesgo la integridad de todo el servicio. Por lo tanto, aplica un paradigma conocido como ejecución limitada, que básicamente determina cuánto tiempo puede ejecutarse su lógica antes de que se agote el tiempo de espera.

Toda la ejecución de la lógica del lado del servidor es completamente transaccional, lo que significa que, si actualiza algunos documentos y se produce un error, o uno de sus scripts se agota debido a una ejecución limitada antes de que realmente se complete, entonces todas las actualizaciones hasta ese momento se revierten automáticamente. Si su código del lado del servidor se completa con éxito, entonces se garantiza que todas las actualizaciones se confirmarán juntas. Esto hace que DocumentDB sea una opción aún más convincente.

ESCALABILIDAD

Con las bases de datos NoSQL, la escalabilidad (horizontal) es clave para el éxito y DocumentDB lo cumple. Es el back-end elegido por servicios como Xbox y Office OneNote, que dependen de DocumentDB para bases de datos que contienen decenas de terabytes de documentos JSON para más de un millón de usuarios y con una disponibilidad de tiempo de actividad del 99,9 %.

Ofrezcamos una idea de cuánto puede escalar DocumentDB: puede crecer tanto como pueda pagar o hasta el final del hardware disponible en los centros de datos de Azure, el límite que alcance primero. Esta es una declaración profunda y una indicación clara de lo que DocumentDB puede manejar y cuánto puede escalar.

Esta es la base utilizada para la creación de Cosmos DB: En términos simples, DocumentDB puede escalar masivamente a cientos de terabytes e incluso petabytes a través de miles de nodos. Puede escalar hacia arriba y hacia abajo y también horizontalmente.

DocumentDB también puede escalar agregando más colecciones⁴. Una colección de documentos puede verse esencialmente como una unidad de escala. Si su base de datos crece más allá de 10 GB, puede escalar horizontalmente simplemente agregando más colecciones y luego dividiendo sus datos en varias colecciones.

⁴ Una colección es, en lenguaje relacional, una base de datos.

CONSISTENCIA

Otra característica importante que hace que DocumentDB sea excelente es su capacidad para ajustar la consistencia. Por lo general, hay una compensación entre el rendimiento y la consistencia.

Básicamente, la consistencia fuerte ralentiza las lecturas y escrituras y la consistencia eventual no siempre devuelve los datos más actuales.

Con una gran consistencia, obtiene resultados de consulta consistentes a medida que los escritores realizan cambios en la base de datos, pero paga un precio en rendimiento ya que todas las consultas deben esperar hasta que todas las réplicas se hayan actualizado con los últimos cambios, lo que obviamente ralentiza un poco las cosas.

Por el contrario, la consistencia eventual le brinda el mejor rendimiento, pero no puede confiar completamente en los resultados de la consulta: Pueden devolver datos que no son del todo coherentes con lo que otros usuarios pueden estar actualizando dado que no todas las réplicas están necesariamente actualizadas.

DocumentDB es compatible con estos dos métodos de consistencia y con tres métodos adicionales que se encuentran en medio de consistencias fuertes y eventuales. Estos tres métodos adicionales se denominan obsolescencia limitada, sesión y prefijo coherente.

La **obsolescencia limitada** le permite tolerar resultados de consulta inconsistentes al garantizar que esos resultados sean al menos lo suficientemente consistentes dentro de un período de tiempo específico.

La **consistencia de la sesión**, que en realidad es el método de consistencia predeterminado que se usa en DocumentDB, se puede considerar como una experiencia híbrida. Se garantiza que los escritores tendrán una gran consistencia para los datos que ellos mismos escribieron, mientras que todos los demás operan con consistencia eventual.

El prefijo consistente garantiza que, en ausencia de más escrituras, las réplicas dentro del grupo finalmente convergen. Las cuentas de Azure Cosmos DB que están configuradas con coherencia de **prefijo coherente** pueden asociar cualquier número de regiones de Azure con su cuenta de Azure Cosmos DB.

DocumentDB permite ajustar y cambiar la consistencia, brindándole la flexibilidad para trabajar con el enfoque que mejor se adapte a los requisitos y necesidades de su negocio.

COSTOS

Antes de la introducción de Cosmos DB, cuando DocumentDB se ofrecía como un servicio independiente, los costos se basaban en niveles y colecciones preestablecidos. En DocumentDB antes de Cosmos DB, una colección no solo era una unidad de escala, sino que también estaba directamente relacionada con los costos y los precios: Se pagaba por colección, y cada colección tenía una capacidad de almacenamiento de hasta 10 GB.

Con la introducción de Azure Cosmos DB, el antiguo esquema de precios basado en la combinación de colecciones y niveles se volvió irrelevante: Los antiguos niveles de rendimiento de DocumentDB S1, S2 y S3 no ofrecían la flexibilidad que ahora ofrecen las colecciones de la API de DocumentDB con Cosmos DB. Esto se debió a que en los niveles de rendimiento S1, S2 y S3, tanto en el rendimiento como la capacidad de almacenamiento, estaban preestablecidos y no ofrecían elasticidad.

Azure Cosmos DB ahora ofrece la capacidad de personalizar su rendimiento y almacenamiento, ofreciéndole mucha más flexibilidad para escalar a medida que cambian las necesidades de su aplicación.

NOTAS SOBRE LAS ESTADÍSTICAS

Esta materia es introductoria a las bases de datos, y, por lo tanto, las estadísticas pueden no ser reveladoras de nada a este nivel de principiantes. Es por eso por lo que rescatamos las conclusiones, ya deglutidas, de expertos en bases de datos que nos señalan qué indican esas estadísticas sobre Cosmos DB:

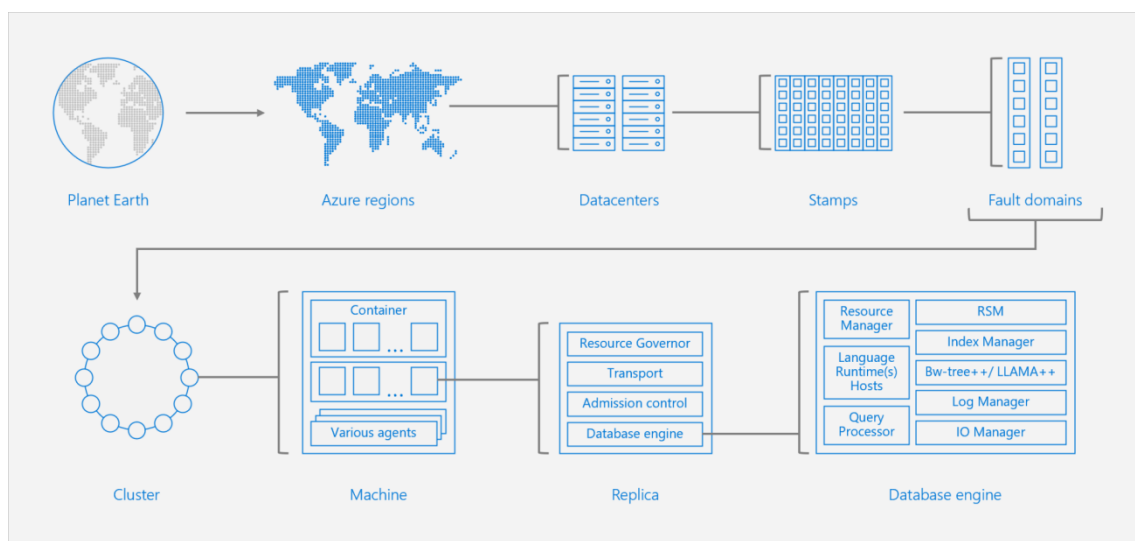
Capacidad de proceso elevada

La estricta semántica de ACID tiene sus ventajas en cuanto a garantizar un estado de datos coherente en la base de datos. Sin embargo, plantea grandes inconvenientes con respecto a la simultaneidad, la latencia y la disponibilidad: Debido a estas restricciones básicas de la arquitectura, los volúmenes transaccionales altos pueden tener como resultado que los datos se deban particionar manualmente.

Ahora bien, la implementación manual de las particiones puede ser un proceso lento y tedioso.

Azure Cosmos DB simplifica estos desafíos ya que se implementa en todo el mundo y en todas las regiones de Azure. Los intervalos de partición se pueden subdividir dinámicamente para aumentar sin problemas el tamaño de la base de datos a la par de la aplicación al tiempo que se mantiene una alta disponibilidad. El gobierno de recursos en la nube multi-inquilino, específico y controlado de forma estricta, facilita sorprendentes garantías de latencia y un rendimiento predecible. La creación de particiones está totalmente administrada, por lo que los administradores no tienen que escribir código ni administrar las particiones.

18

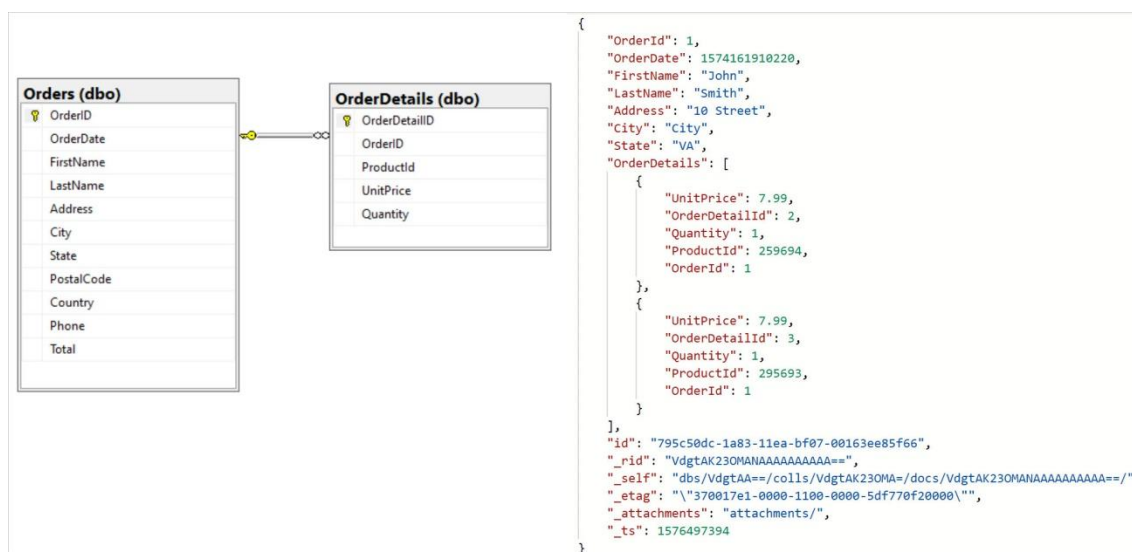


Datos jerárquicos

Las bases de datos NoSQL podrían pensarse como una actualización del paradigma jerárquico de base de datos (paradigma en boga en los '80) pero con la diferencia de que ahora no se ven obstaculizadas por el costo de almacenar los datos en el disco.

Como resultado, el mantenimiento de muchas relaciones entidades complejas entre elementos primarios y secundarios en una base de datos relacional se puede considerar ahora un antipatrón en comparación con los métodos modernos orientados a documentos: Con estos métodos, los desarrolladores no se ven obligados a comprometerse con los controladores de mapeo objeto-relacional ni con los motores de base de datos orientados a objetos.

Es decir: Si los datos contienen muchas relaciones de elementos primarios y secundarios y jerarquías internas, la API de SQL de Azure Cosmos DB ofrece una mejor solución.



19

Redes y relaciones complejas

Las relaciones entre entidades no existen realmente en una base de datos relacional. Deben calcularse en tiempo de ejecución.

Las relaciones complejas requieren combinaciones cartesianas para permitir la asignación mediante consultas, y como resultado, las operaciones de cálculo se vuelven más costosas a medida que aumentan dichas relaciones.

Esto, a lo menos, resulta paradójico ya que las bases de datos relacionales presentan una solución poco óptima para el modelado de relaciones profundas y complejas, a pesar de su nombre.

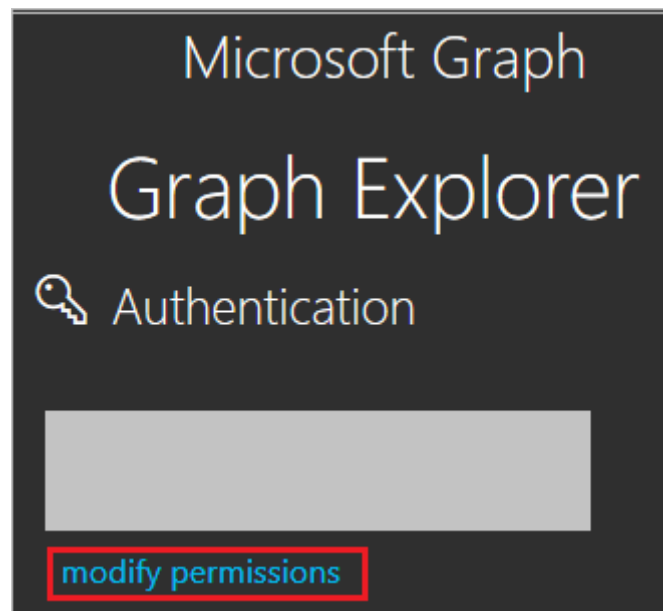
Junto con el modelo relacional de Ted Codd, surgieron otros modelos en forma paralela. Uno, fue el jerárquico (citado anteriormente) y otro fue el de bases de datos de red. Pero ambos sistemas no ganaron popularidad por la falta de casos de uso y la ineficacia de almacenamiento para estos modelos.

Los motores de base de datos de grafos podrían considerarse como la reaparición del paradigma de la base de datos de red con una clara ventaja: son sistemas basados primariamente en las relaciones, por lo cual, las relaciones se pueden recorrer en sin que la complejidad temporal aumente con cada nueva combinación.

Azure Cosmos DB es un servicio de base de datos con varios modelos, que ofrece una proyección de API para todos los tipos de modelos NoSQL principales: familia de columnas, documentos, grafos y pares clave-valor. Las capas de la API de documentos de Gremlin (grafo) y SQL (Core) son completamente interoperables.

Por tanto, si se piensa mantener una red compleja de relaciones en la base de datos, una base de datos de grafos como la API de Gremlin de Azure Cosmos DB sería una solución más que adecuada.

20



Esquema fluido

La característica más prominente concreta del modelo relacional es que los esquemas se deben definir durante el tiempo de diseño a fin de explotar las ventajas que ofrece el modelo en cuanto a las ventajas de la integridad referencial y la conformidad (coherencia) de los datos, lo cual también, como contraparte, es restrictivo a la hora de responder a cambios a medida que crece la aplicación.

Cuando el nivel de complejidad en adoptar los cambios que el crecimiento de la aplicación requiere, es claro el beneficio de que el esquema se transfiera a la aplicación para administrarse por registro.

Esto implica, ciertamente, dos cosas: (1) la base de datos debe ser independiente del esquema y (2) debe permitir que los registros sean "autodescriptivos" (en cuanto a los datos contenidos en ellos).

Por tanto, si va a administrar datos cuyas estructuras cambian constantemente, Azure Cosmos DB es una solución óptima.

Microservicios

21

El patrón de microservicios ha crecido significativamente en los últimos años. Este patrón tiene sus orígenes en la arquitectura orientada a servicios. El estándar de facto para la transmisión de datos en estas arquitecturas de microservicios modernas es JSON, que también es el medio de almacenamiento para la inmensa mayoría de las bases de datos NoSQL orientadas a documentos. Esto hace que los almacenes de documentos NoSQL sean opciones mucho mejores para la persistencia y la sincronización (mediante patrones de abastecimiento de eventos) en implementaciones de microservicios complejas. Las bases de datos relacionales más tradicionales pueden tener un mantenimiento mucho más complejo en estas arquitecturas. Esto se debe a que se necesita una mayor cantidad de transformaciones para los estados y la sincronización entre las API. En concreto, Azure Cosmos DB tiene una serie de características que lo convierten en una opción aún mejor para las arquitecturas de microservicios basadas en JSON que muchas bases de datos NoSQL:

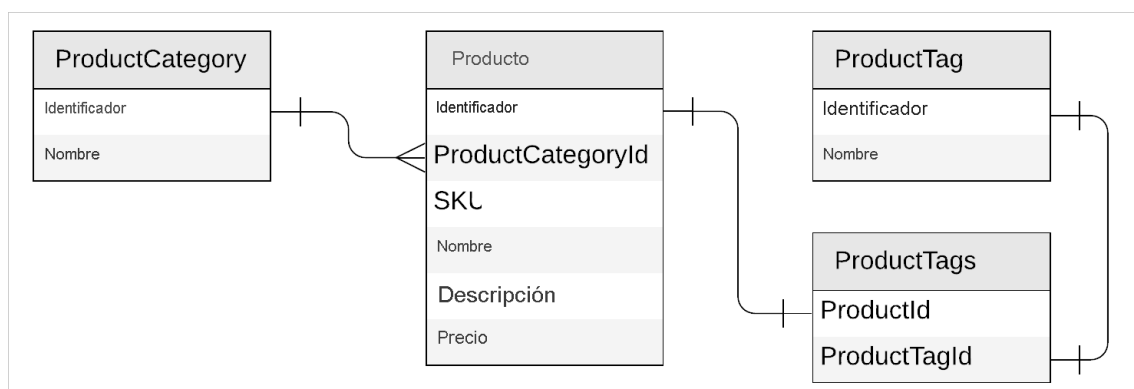
- selección de tipos de datos JSON puros;
- un motor de JavaScript y una API de consulta integradas en la base de datos;
- una fuente de cambios vanguardista a la que los clientes pueden suscribirse para recibir notificaciones sobre las modificaciones realizadas en un contenedor.

Consideraciones críticas

Hay aspectos que implican algún grado de dificultad presentes en el diseño relacional. Pero esas mismas dificultades, en un modelo NoSQL, ven incrementada la complejidad de implementación.

TRANSACCIONES CON VARIAS RELACIONES QUE APUNTAN A LA MISMA ENTIDAD

La regla general de las bases de datos NoSQL suele ser la desnormalización (la cual genera lecturas más eficaces en un sistema distribuido). Sin embargo, hay algunos desafíos de diseño que entran en juego con este método. El siguiente es un ejemplo de un producto relacionado tanto con una categoría y como con varias etiquetas:



Un método recomendado en una base de datos de documentos NoSQL sería desnormalizar el nombre de la categoría y los nombres de las etiquetas directamente en un "documento de producto". Sin embargo, para mantener sincronizadas las categorías, las etiquetas y los productos, las opciones de diseño que facilitan esta tarea, por el contrario, han agregado complejidad al mantenimiento ya que los datos se duplican en varios registros del producto (en lugar de ser una sencilla actualización de una relación de "uno a varios" de un modelo relacional).

TRANSACCIONES QUE NECESITAN UNA GRAN COHERENCIA EN TODO EL CONJUNTO DE DATOS

Es poco habitual que sea en un modelo NoSQL sea necesario un alto nivel de coherencia en todo el conjunto de datos. Sin embargo, si se diera el caso, sería un desafío enorme en el caso de las bases de datos distribuidas porque, para garantizar dicha coherencia, se deben sincronizar los datos en todas las réplicas y regiones antes de permitir la lectura de los clientes. Esto, inevitablemente, aumentará la latencia de todas las lecturas.

NOTAS SOBRE UX (EXPERIENCIA DE USUARIO)

En el caso de las bases de datos, ¿quién es el usuario final? Permítasenos arriesgar que el usuario de las bases de datos, por lo general, no es el usuario final de las aplicaciones que ofrecemos como programadores. Probablemente seamos nosotros, los programadores, uno de los grupos de usuarios que interactúa con una base de datos. O al menos mientras dure la etapa de desarrollo.

Cuando quisimos indagar sobre la experiencia de usuario en Cosmos DB, la respuesta que más usualmente encontramos es la del punto de vista de administradores de sistema o administradores de bases de datos. Y está bien, pero para nuestro nivel principiante además orientado a la programación, esas apreciaciones, la de ellos, parecen artes oscuras difíciles de descifrar. Pero de lo que sí sabemos es cómo interactuar con una base de datos, cualquiera, cuando desarrollamos nuestras aplicaciones. Y entre nosotros comentamos, ya se sabe: ¿es fácil? ¿es portable? ¿demanda muchos recursos? ¿interactúa bien con mi IDE? Etcétera.

Esta parte, con la que cerramos el trabajo, lo hacemos a manera de un documental para responder a la pregunta: ¿qué tal es, como programadores, trabajar con Cosmos DB?

23

Primeros pasos (el Data-End)

Cosmos DB es parte de la familia de Azure, la solución en la nube de Microsoft. Paga. Y algunos dicen que no tan barata. A un desarrollador eso lo desanima, claro está. Pero entre la gente de Microsoft hay desarrolladores. O al menos, eso creemos. Y creemos que entienden los vericuetos que los programadores enfrentamos. Y creemos que, por esa razón, Microsoft ofrece un emulador de Cosmos DB (y realmente es un emulador, ya se verá) para que uno pueda “jugar” con Cosmos DB hasta el día que uno se ponga serio, madure, y adquiera una suscripción (como todo adulto responsable).

Todo lo que hay que saber sobre la descarga, instalación y funcionamiento del emulador puede ser consultado en <https://docs.microsoft.com/es-es/azure/cosmos-db/local-emulator> y por eso no nos detendremos en este punto porque no es lo que queremos compartir. Queremos relatar cómo es la experiencia de interactuar con Cosmos DB.

Ya una vez instalado el emulador (no necesita configuración alguna de nuestra parte, todo está automatizado por el instalador) veremos lo siguiente:

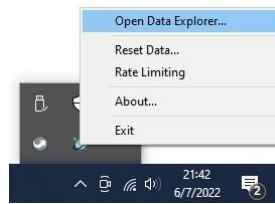


Ilustración 1

En el escritorio de Windows, barra de tareas, la forma de arrancar el servidor local (esto de servidor es justamente eso: salvo algunas funciones, el emulador nos da una visión real de lo que hay en Azure) es hacerlo a través del menú contextual del ícono de Cosmos DB.

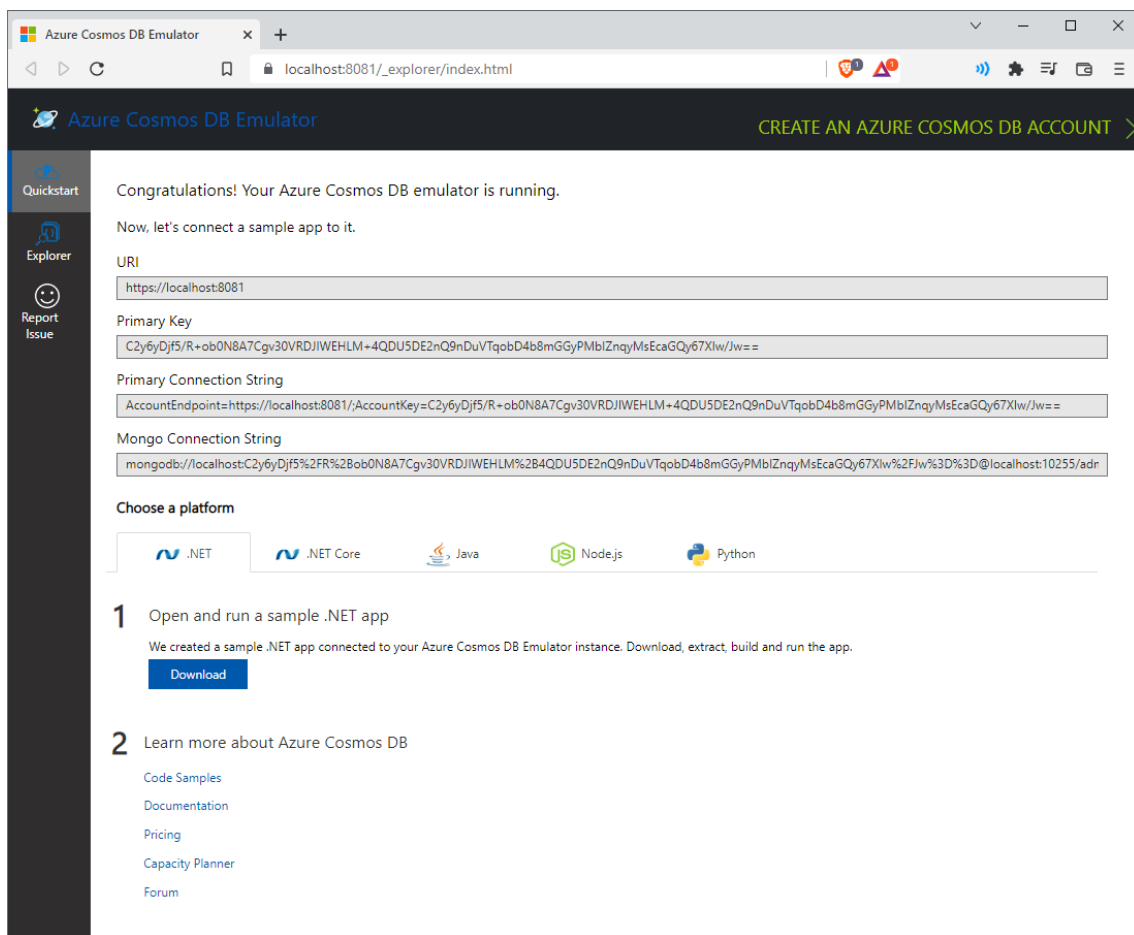


Ilustración 2

El emulador abrirá el Panel de Control de Cosmos DB, el que (por obvias razones) abre una cuenta habilitada y con los elementos necesarios (URI y Clave) para empezar. Es decir, como comentábamos anteriormente, completamente automatizado y listo para usar.

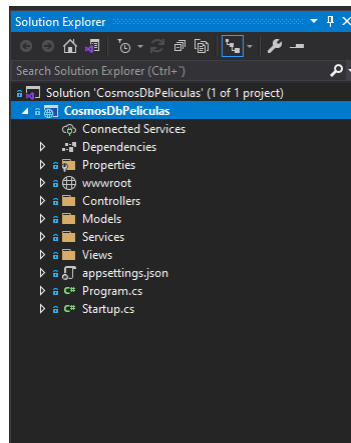


Ilustración 3

Quien haya encarado alguna vez un proyecto básico de ASP.NET, va a reconocer esta estructura de archivos en Visual Studio. No hace falta agregar ni quitar nada, todo queda como está.

Congratulations! Your Azure Cosmos DB emulator is running.

Now, let's connect a sample app to it.

URI

`https://localhost:8081`

Primary Key

`C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XlIw/Jw==`

Primary Connection String

`AccountEndpoint=https://localhost:8081/;AccountKey=C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XlIw/Jw==`

Mongo Connection String

`mongodb://localhost:C2y6yDjf5%2FR%2Bob0N8A7Cgv30VRDJIWEHLM%2B4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XlIw%2FJw%3D%3D@localhost:10255/admin?ssl=true`

25

Volvamos al Panel de Control de Cosmos DB. Como se ve, se nos provee (sin necesidad de configurar nada) tanto la URI como la Primary Key. Debemos copiarla y llevarla al archivo `appsettings.json` de nuestro proyecto.

```

1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft": "Warning",
6        "Microsoft.Hosting.Lifetime": "Information"
7      }
8    },
9    "AllowedHosts": "*",
10   "CosmosDb": {
11     "endPoint": "https://localhost:8081",
12     "primaryKey": "C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XlIw/Jw=="
13   }
14 }
15

```

Ilustración 4

Por el lado del servidor, hemos terminado.

El desarrollo (Back-End & Front-End)

```
smosDbPelículas
1 using Newtonsoft.Json;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Threading.Tasks;
6
7 namespace CosmosDbPelículas.Models
8 {
9     public class Película
10     {
11         [JsonProperty(PropertyName = "id")]
12         public String Id { get; set; }
13         public String Categoria { get; set; }
14         public String Titulo { get; set; }
15         public String Director { get; set; }
16         public String Estreno { get; set; }
17
18         public override string ToString()
19         {
20             return JsonConvert.SerializeObject(this);
21         }
22     }
23 }
24
```

Ilustración 5

Ya en el código, creamos nuestro modelo «Película» (que, además, como un plus, tiene un método de serialización).

```
smosDbPelículas
70 Película película = JsonConvert.DeserializeObject<Película>(await stream.ReadToEndAsync());
71 return película;
72 }
73
74 public async Task ModificarPelícula(Película película)
75 {
76     Uri uri = UriFactory.CreateDocumentUri(this.bbdd, this.collection, película.Id);
77     await this.client.ReplaceDocumentAsync(uri, película);
78 }
79
80 public async Task EliminarPelícula(String id)
81 {
82     Uri uri = UriFactory.CreateDocumentUri(this.bbdd, this.collection, id);
83     await this.client.DeleteDocumentAsync(uri);
84 }
85
86 public List<Película> BuscarPelículas(String categoria)
87 {
88     FeedOptions options = new FeedOptions() { MaxItemCount = -1 };
89     Uri uri = UriFactory.CreateDocumentCollectionUri(this.bbdd, this.collection);
90     String sql = "select * from c where c.Categoria='" + categoria + "'";
91     IQueryable<Película> query = this.client.CreateDocumentQuery<Película>(uri, sql, options);
92     IQueryable<Película> queryLambda = this.client.CreateDocumentQuery<Película>(uri, options)
93         .Where(z => z.Categoria == categoria);
94
95     return query.ToList();
96 }
97
```

Ilustración 6

Y a continuación, creamos nuestro servicio. Como se puede ver, es nuestro viejo y querido amigo, el ABM. Nótese que la consulta (en rojo) es ...SQL (SQL común y silvestre).

Podríamos seguir mostrando un par de pantallas más de código, pero no hay nada de novedoso pues son ventanas usuales, de forma, demandadas por ASP.NET para un CRUD (Controlador y Vista). Así que saltamos estos dos pasos y vamos al tercero y último: las dependencias (y con eso, se terminó la etapa de desarrollo).

```
1 using CosmosDbPeliculas.Services;
2 using Microsoft.AspNetCore.Builder;
3 using Microsoft.AspNetCore.Hosting;
4 using Microsoft.Extensions.Configuration;
5 using Microsoft.Extensions.DependencyInjection;
6 using Microsoft.Extensions.Hosting;
7
8 namespace CosmosDbPeliculas
9 {
10     public class Startup
11     {
12         public Startup(IConfiguration configuration)
13         {
14             Configuration = configuration;
15         }
16
17         public IConfiguration Configuration { get; }
18
19         // This method gets called by the runtime. Use this method to add services to the container.
20         public void ConfigureServices(IServiceCollection services)
21         {
22             services.AddTransient<ServiceCosmosDb>();
23             services.AddControllersWithViews();
24         }
25
26         // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
27         public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
28         {
29             if (env.IsDevelopment())
```

Ilustración 7

¿Por qué ponemos esta última pantalla, si casi todo el código que hemos mostrado hasta ahora es prácticamente el que viene por defecto en cualquier plantilla de Visual Studio para un proyecto de ASP.NET?

Justamente por eso: como puede apreciarse, pusimos

- la cadena de conexión (Ilustración 4),
- definimos nuestro modelo de objetos (Ilustración 5),
- declaramos sucintamente nuestro ABM (Ilustración 6)

...y eso fue todo.

Para nuestra demostración, es todo el código que escribiremos.

La aplicación

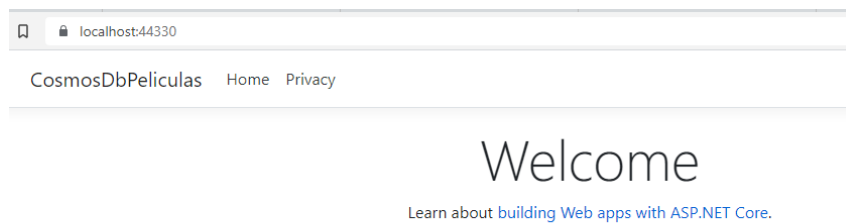


Ilustración 8

Iniciamos la prueba en DEBUG de la aplicación y podemos ver la primera pantalla. Hasta el momento, todo bien.

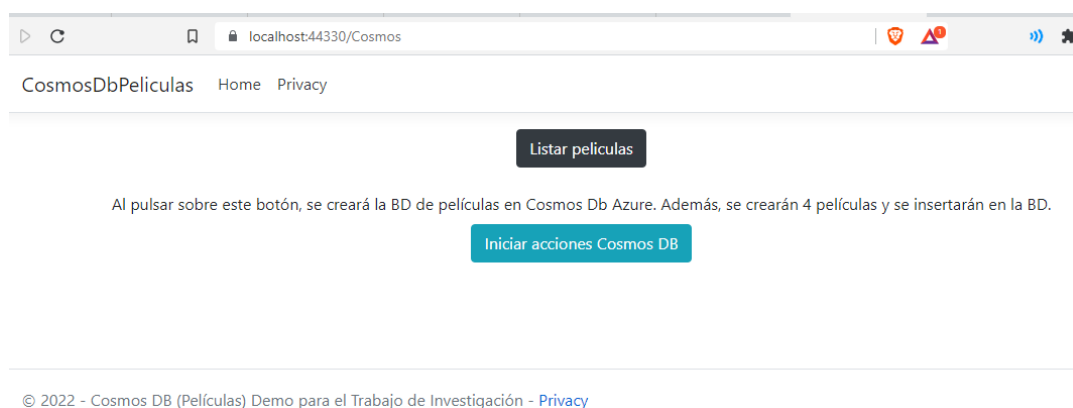


Ilustración 9

El «Iniciar Acciones» fue algo que se agregó a los fines de este demo. Ahora comienza la interacción con Cosmos DB.

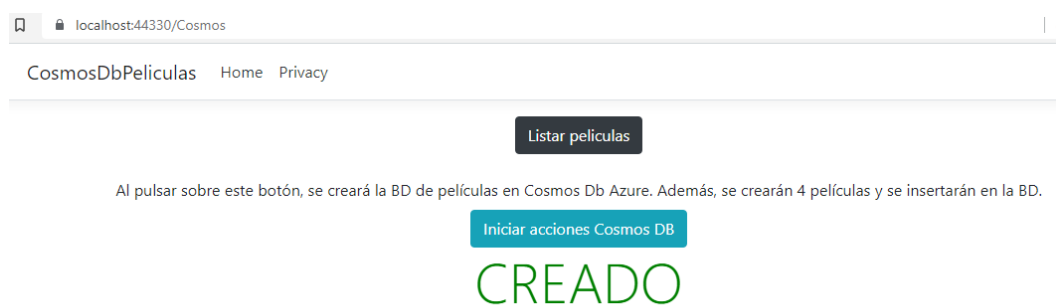
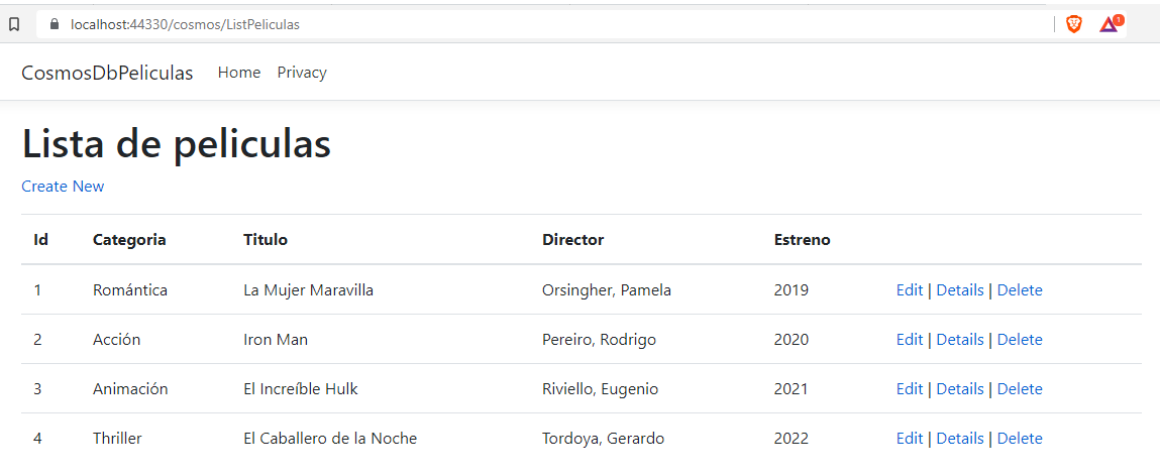


Ilustración 10

La puesta en escena tardó menos de 1 minuto. Ya está todo listo: aplicación y base de datos operativas.

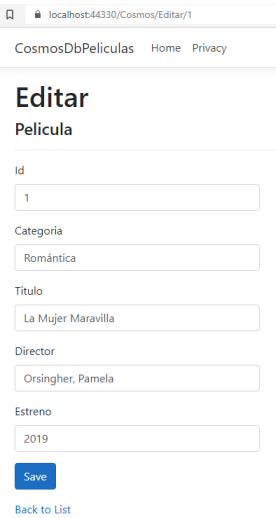
Pasos finales (UX)



CosmosDbPelículas Home Privacy					
Lista de películas					
Create New					
Id	Categoría	Título	Director	Estreno	
1	Romántica	La Mujer Maravilla	Orsingher, Pamela	2019	Edit Details Delete
2	Acción	Iron Man	Pereiro, Rodrigo	2020	Edit Details Delete
3	Animación	El Increíble Hulk	Riviello, Eugenio	2021	Edit Details Delete
4	Thriller	El Caballero de la Noche	Tordoya, Gerardo	2022	Edit Details Delete

Ilustración 11

La aplicación, en pantalla, nos muestra lo que buscábamos: un ABM bien dibujado. Básico, pero funcional. Veamos las pantallas de Edit y Details.



localhost:44330/Cosmos/Editar/1

CosmosDbPelículas Home Privacy

Editar Película

Id

Categoría

Título

Director

Estreno

[Save](#)

[Back to List](#)

Ilustración 12



Ilustración 13

Y ahora veamos el Panel de Control de Cosmos DB, que imagino es lo que en definitiva interesa (ya habiendo dejado en claro los estamentos de la interacción).

La frutilla del postre

30

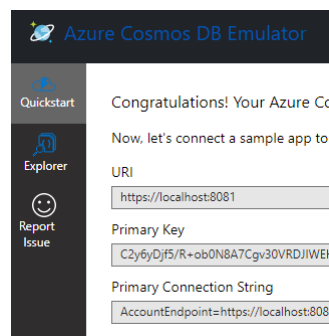


Ilustración 14

Veamos con más detalle el menú del Panel de Control de Cosmos DB. Ya lo vimos, pero volvamos sobre su simplicidad. Tres pantallas. La última, de contacto, para Microsoft. La primera, lo que necesitamos para nuestra conexión. Ahora, la segunda, el Explorer, nos da todo lo necesario para la administración de Cosmos DB. Veamos.

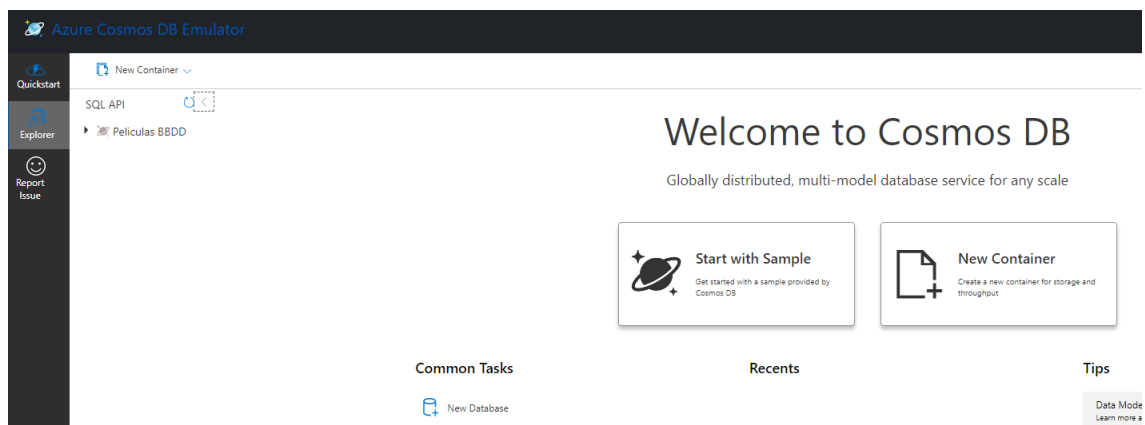


Ilustración 15

Como puede verse, “habemus data”. Veamos qué tenemos en nuestra recién nacida base de datos de película.

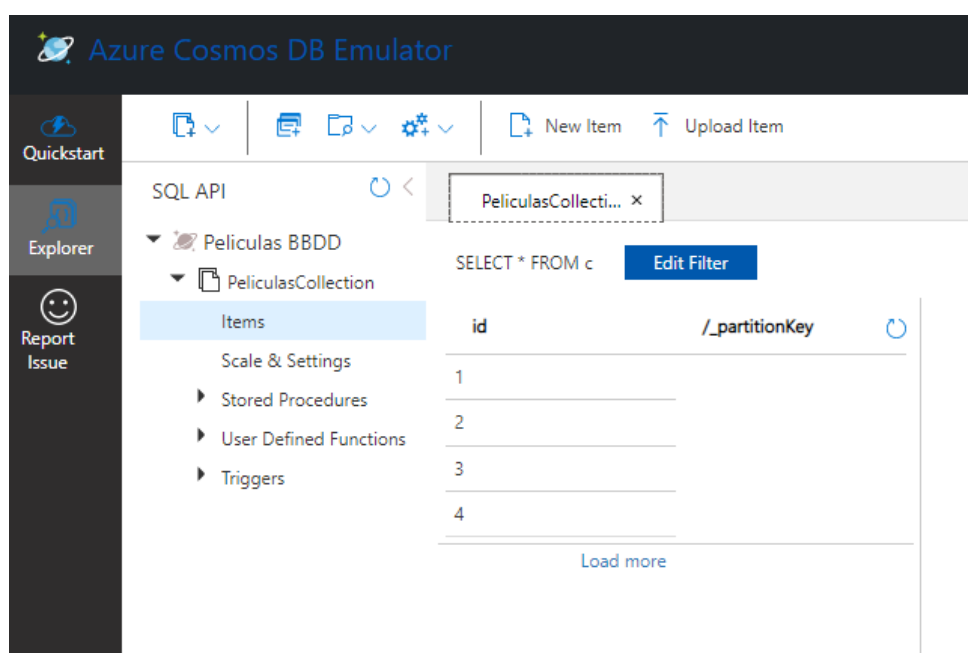


Ilustración 16

¿Se acuerdan de que comentábamos de ciertos conceptos familiares en las bases de datos relacionales también presentes en Cosmos DB? Pues bien, de los creadores de SQL Server y SSMS, les presentamos lo que obtienen con Cosmos DB. Cualquier parecido con la realidad, no es mera coincidencia, es así.

No nos queremos alargar. Hemos tratado de ser lo más sucintos posible, pero convengamos que hay mucho que ver. Solo cerremos esta parte mostrando el famoso JSON del que hablamos a lo largo de todo este trabajo, y cuando lo haga, piense en todo el camino que hizo ese JSON, todo ese

trabajo, todas esas mentes, todas esas tecnologías, todos esos acuerdos entre miles de desarrolladores, entre cientos de propuestas, para llegar a la simplicidad y la belleza de un formato simple de intercambio universal, por qué no.

SELECT * FROM c

Edit Filter

id	/_partitionKey
1	
2	
3	
4	

Load more

```
1 {
2   "id": "1",
3   "Categoria": "Romántica",
4   "Titulo": "La Mujer Maravilla",
5   "Director": "Orsingher, Pamela",
6   "Estreno": "2019",
7   "_rid": "o1BNAON1QhsBAAAAAAAAA==",
8   "_self": "dbs/o1BNAA==/colls/o1BNAON1Qhs=/docs/o1BNAON1QhsBAAAAAAAAA==/",
9   "_etag": "\"00000000-0000-0000-91a8-77e2c2e001d8\"",
10  "_attachments": "attachments/",
11  "_ts": 1657160575
12 }
```

Ilustración 17

CONCLUSIÓN

Selección de la opción de API ...

¿Qué API resulta más adecuada para la carga de trabajo?

Azure Cosmos DB es un servicio de base de datos NoSQL totalmente administrado para la creación de aplicaciones escalables y de alto rendimiento. [Learn more](#)

Para empezar, seleccione la API para crear una cuenta nueva. La selección de la API no se puede cambiar tras crear la cuenta.

Núcleo (SQL): opción recomendada
Núcleo o API nativa de Azure Cosmos DB para trabajar con documentos. Permite un desarrollo rápido y flexible con el lenguaje de consultas SQL y las bibliotecas cliente para .NET, JavaScript, Python y Java que ya conoce.

API de Azure Cosmos DB para MongoDB
Servicio de base de datos de MongoDB totalmente administrado para aplicaciones escritas para MongoDB. Se recomienda si dispone de cargas de trabajo existentes de MongoDB que tenga previsto migrar a Azure Cosmos DB.

Cassandra
Servicio de base de datos de Cassandra totalmente administrado para aplicaciones escritas para Apache Cassandra. Se recomienda si dispone de cargas de trabajo existentes de Cassandra que tenga previsto migrar a Azure Cosmos DB.

Tabla de Azure
Servicio de base de datos totalmente administrado para aplicaciones escritas para el almacenamiento de Azure Table. Se recomienda si dispone de cargas de trabajo existentes del almacenamiento de Azure Table que tenga previsto migrar a Azure Cosmos DB, pero no quiere volver a escribir la aplicación para usar la API SQL.

Gremlin (grafo)
Servicio de base de datos de grafos totalmente administrado que emplea el lenguaje de consultas Gremlin, basado en el proyecto Apache TinkerPop. Se recomienda para nuevas cargas de trabajo en las que sea necesario almacenar relaciones entre datos.

Ilustración 18⁵

Si debemos ir al grano, al hueso mismo, podríamos decir lo siguiente: si a lo largo de la materia no hubiéramos abordado y considerado los conceptos del modelo relacional, sus bondades y sus exigencias; si nos hubiéramos limitado a encarar lo NoSQL de buenas a primera sin tener nociones previas bien estudiadas, lo más probable es que no hubiéramos apreciado lo que ofrece NoSQL y no le hubiéramos visto ni valor ni utilidad alguna.

Tal vez estemos errados, pero de aquí en más pensamos que la forma de encarar el estudio de NoSQL es primero entender el modelo relacional. Luego entender los pros y los contras de la propuesta NoSQL. Y ahí, recién ahí, poder sentarnos con tranquilidad y evaluar si Cosmos DB, con su propuesta multimodal no termina siendo una de las propuestas más atractivas de la informática en su estado del arte actual.

Gracias por su atención.

Link de video: <https://www.youtube.com/watch?v=d8Z2EhqQcRo>

⁵ ¿Por qué de esta última imagen? ¿Qué intentamos decir? Que Cosmos DB es una “navaja suiza” que bien puede acomodarse con flexibilidad a distintos tipos de proyectos usando una misma herramienta. Decimos desarrollo, ¿nos arriesgamos a proponerla como herramienta de aprendizaje? ¿Por qué no?

FUENTES (BIBLIOGRAFÍA)

Syncfusion Technology Resource Portal (<https://www.syncfusion.com/>)

Documentación Técnica de Microsoft (<https://docs.microsoft.com/>)

TechClub Tajamar (<https://techclub.tajamar.es/>)